Thesis for the degree of Licentiate of Engineering

Tactical decision-making for autonomous driving: A reinforcement learning approach

CARL-JOHAN HOEL

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Tactical decision-making for autonomous driving:
A reinforcement learning approach
CARL-JOHAN HOEL

THESIS FOR LICENTIATE OF ENGINEERING no 2019:07

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE–412 96 Gothenburg
Sweden
Telephone: +46 (0)31–772 1000

Chalmers Reproservice
Gothenburg, Sweden 2019

Tactical decision-making for autonomous driving: A reinforcement learning approach

Carl-Johan Hoel
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

The tactical decision-making task of an autonomous vehicle is challenging, due to the diversity of the environments the vehicle operates in, the uncertainty in the sensor information, and the complex interaction with other road users. This thesis introduces and compares three general approaches, based on reinforcement learning, to creating a tactical decision-making agent. The first method uses a genetic algorithm to automatically generate a rule based decision-making agent, whereas the second method is based on a Deep Q-Network agent. The third method combines the concepts of planning and learning, in the form of Monte Carlo tree search and deep reinforcement learning. The three approaches are applied to several highway driving cases in a simulated environment and outperform a commonly used baseline model by taking decisions that allow the vehicle to navigate 5% to 10% faster through dense traffic. However, the main advantage of the methods is their generality, which is indicated by applying them to conceptually different driving cases. Furthermore, this thesis introduces a novel way of applying a convolutional neural network architecture to a high level state description of interchangeable objects, which speeds up the learning process and eliminates all collisions in the test cases.

Keywords: Autonomous driving, tactical decision-making, deep reinforcement learning, neural network, Monte Carlo tree search, genetic algorithm.

Till min lilla trollunge.

Acknowledgements

I would first like to express gratitude to my supervisors Prof. Krister Wolff and Prof. Leo Laine, for believing in me and giving me guidance on this journey. I am also grateful to my examiner Prof. Mattias Wahde for taking me in as a PhD student and for helping me to improve my scientific writing skills.

I would also like to thank my colleagues at Volvo, both at the Traffic situation management group and the Vehicle analysis group, and at Chalmers at the VEAS division, for interesting discussions and a friendly working environment. My new colleagues at AI Innovation of Sweden have already given me a lot of inspiration and I am looking forward to find out what the future will bring for the center. I would also like to thank Prof. Mykel Kochenderfer and Prof. Katie Driggs-Campbell at Stanford for welcoming me to SISL for a couple of months, which gave me some valuable insights and new perspectives.

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP), funded by Knut and Alice Wallenberg Foundation, and partially by Vinnova FFI.

List of included papers

This thesis consists of the following papers, in all cases with the author of the thesis as the main contributor. References to the papers will be made using Roman numerals.

I. C. J. Hoel, M. Wahde, and K. Wolff, *An evolutionary approach to general-purpose automated speed and lane change behavior.* In: Proceedings of the 16$^{\text{th}}$ IEEE International Conference on Machine Learning and Applications, Cancun, Mexico, 2017, pp. 743-748.

II. C. J. Hoel, K. Wolff, and L. Laine, *Automated speed and lane change decision making using deep reinforcement learning.* In: Proceedings of the 21$^{\text{st}}$ IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 2018, pp. 2148-2155.

III. C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, *Combining planning and deep reinforcement learning in tactical decision making for autonomous driving.* Submitted to: IEEE Transactions on Intelligent Vehicles, 2019.

Table of Contents

Chapter 1

Introduction

Autonomous vehicles are expected to have a radical effect on the transportation system, and the technology has the potential to bring many benefits to society [9]. For example, the traffic safety is expected to increase, since the majority of accidents are currently caused by human factors. Furthermore, transportation of goods could be done during the night, which would reduce congestion, and the productivity of commercial vehicles will increase when fewer human drivers are needed. Autonomous vehicles will also be able to drive in a more energy efficient way than vehicles that are operated by humans.

The decision-making task of an autonomous vehicle can be divided into three different levels: strategic, tactical, and operational decision-making [17], also referred to as navigation, guidance, and stabilization [30]. The strategic level considers the high level goals of a trip and handles the route planning, whereas the tactical decision-making level modifies the strategic plan in order to adapt to the current traffic situation. The tactical decisions could for example consider when to change lanes, or whether or not to stop at an intersection. Finally, the operational decision-making level translates the maneuvers of the tactical level into control operations of the vehicle. The topic of this thesis is the tactical decision-making level.

To create a tactical decision-making agent for autonomous driving is a challenging task, since the agent has to base its decisions on information with a varying degree of uncertainty. Sensor imperfections and occlusions make the current state of a traffic scene uncertain. The future development of the scene is also hard to predict, since it depends on the intentions of

other road users and on how the agent interacts with them. Furthermore, an autonomous vehicle will encounter a varied set of environments, which could range from structured highway driving to turbulent urban environments. To anticipate all possible traffic situations that a vehicle should be able to handle, and manually code a suitable behavior for these, would be extremely time consuming and error prone, if at all possible.

Many methods for tactical decision-making already exist; an overview is given in Chapter 2. A common limitation for most of them is that they are designed for a particular driving case. To take a method that was designed for, e.g., a highway driving case and transfer it to a crossing case is, if at all possible, not straightforward. Therefore, an autonomous vehicle that operates in different environments typically needs to use a large number of different methods to solve different driving cases, which can be both impractical and difficult to manage.

This thesis introduces and analyzes three different approaches to creating a tactical decisions making agent, which are all based on reinforcement learning (RL) methods [27]. The three approaches are mainly tested on different highway driving cases, and the agents are trained in a simulated environment. The first approach uses a genetic algorithm (GA) [11] to automatically generate a rule based decision-making agent, whereas the second approach is based on a Deep Q-Network (DQN) agent [18]. The second approach also introduces a new way of applying a convolutional neural network architecture [15] to a high level state description of interchangeable objects, which significantly improves both the training speed and the quality of the final agent. The third approach incorporates more domain knowledge and combines the concepts of planning and learning, in the form of Monte Carlo tree search (MCTS) [6] and deep reinforcement learning. This method is inspired by the AlphaGo Zero algorithm [24], which is here first extended to a domain with a continuous state space and where self-play cannot be used, and then adopted to the autonomous driving domain. All approaches outperform a commonly used baseline method by navigating 5% to 10% faster through dense traffic. The strength of combining planning and learning in the third approach is also illustrated by comparing it to using the planning or the learned policy separately. The main benefit of the three approaches is that they are general, i.e., not limited to a specific driving case. See Chapter 2 for a further comparison with contemporary approaches.

1.1 Contributions

The main contributions of this thesis are:

- The introduction of three conceptually general approaches to creating a tactical decision-making agent, which all show promising results by outperforming a commonly used baseline model in different highway driving cases (Chapter 4 and Paper I, II, III).
- A comparison and analysis of the properties of the three approaches (Chapter 5).
- The introduction of a novel way of using a convolutional neural network architecture, which speeds up the training process and improves the quality of the trained agent (Section 4.4 and Paper II).
- An extension of the AlphaGo Zero algorithm, which allows it to be used as a framework for tactical decision-making in the autonomous driving domain (Section 4.5 and Paper III).

1.2 Thesis outline

A review of related work is presented in Chapter 2, which is followed by a brief background to Markov decisions processes (MDPs) and reinforcement learning in Chapter 3. This chapter also introduces notation and terminology that are used in the subsequent chapters. Chapter 4 presents the test cases of the agents and how the decision-making problem is formulated as a partially observable Markov decisions process (POMDP), followed by a description of the three different RL approaches and a presentation of the main results of the trained agents. The properties of the three approaches are discussed and compared in Chapter 5. Finally, Chapter 6 provides concluding remarks and future research directions.

Chapter 2

Related Work

Early approaches to tactical decision-making for autonomous vehicles often used rule-based methods, commonly implemented as handcrafted state machines. For example, during the DARPA Urban Challenge, this method was adopted by the winning team from the Carnegie Mellon University, where different modules handled the behavior for the different driving cases that were encountered [33]. Other participants, such as Stanford and Virginia Tech, used similar strategies [19], [3]. However, these rule-based approaches lack the ability to generalize to unknown situations. Furthermore, they do not deal with input uncertainties.

Another group of algorithms treats the decision-making task as a motion planning problem. Commonly, a prediction model is used to predict the motion of the other agents, and then the behavior of the vehicle that is being controlled, henceforth referred to as the ego vehicle, is planned accordingly. This results in a reactive behavior, since the predictions are independent of the ego vehicle plan. Therefore, interaction between the ego vehicle and other agents is not explicitly considered, but may happen implicitly by frequent replanning. A motion planning approach for a highway scenario is for example used by Werling et al. [35] and Nilsson et al. [21]. Since human behavior is complex and varies between individuals, some algorithms use a probabilistic prediction as input to the motion planning. This is for example shown in a study by Damerow et al. [8], which aims to minimize the risk during an intersection scenario.

It is common to model decision-making problems as partially observable Markov decision processes [14], see Section 3.1 for more details. This

mathematical framework allows modeling of uncertainty in the current state, uncertainty in the future development of the traffic scene, and modeling of an interactive behavior. The task of finding the optimal policy for a POMDP is difficult, but many approximate methods exist. One way to group these methods is in offline and online methods. There are powerful offline algorithms for planning in POMDPs, which can solve complex situations. One example is shown by Brechtel et al., which proposes a solution to how measurement uncertainty and occlusions in an intersection can be handled. An offline planner precomputes the policy by using a state representation that is learned for the specific scenario [5]. A similar approach is adopted by Bai et al. for an intersection scenario [4]. The main drawback of these offline methods is that they are designed for specific scenarios. Due to the large number of possible real world scenarios, it becomes impossible to precalculate a policy that is generally valid.

Online methods compute a policy during execution, which makes them more versatile than offline methods. However, the limited available computational resources requires a careful problem formulation and limits the solution quality. Ulbrich et al. [31] use a POMDP framework to make decisions on lane changes during highway driving. In order to make it possible to solve the problem with an exhaustive search, a problem-specific high level state space is created, which consists of states that represents whether or not a lane change is possible or beneficial. However, due to the specialized state space, it is hard to generalize this method. Another online method for solving a POMDP is Monte Carlo tree search [6]. Sunberg et al. use MCTS to make decisions for changing lanes on a highway [26]. In order to handle the continuous state description, the tree search is extended with a technique called progressive widening [7]. Furthermore, other drivers' intentions are estimated with a particle filter. A hybrid approach between offline and online planning is pursued in a study by Sonu et al., where a hierarchical decision-making structure is used. The decision-making problem is modeled on two levels as MDPs, since full observability is assumed. The high level MDP is solved offline by value iteration and the low level MDP is solved online with MCTS [25].

Reinforcement learning methods are versatile, and have proved successful in various domains, such as playing Atari games [18], in continuous control [16], reaching a super human performance in the game of Go [24], and beating the best chess computers [23]. One advantage of using RL methods for solving POMDPs is that they can be model-free, i.e., the transition

probabilities between different states do not need to be known or modeled. Furthermore, many RL methods are general and an agent could, in theory, learn how to behave correctly in all possible driving situations. A Deep Q-Network agent that can learn how to negotiate an intersection, while interacting with drivers with different intentions, is presented by Tram et al. [28]. This approach uses a high level state description and action space. A DQN agent is also used by Isele et al., who consider several different intersection scenarios, but uses a discretized bird's eye view description of the state, which can also represent occluded areas [12]. Expert knowledge that restrict certain actions can be utilized, which can speed up the training process for a lane changing scenario [20]. A different approach is presented by Shalev et al., applied to a merging scenario on a highway, where a policy gradient method is used to learn a desired behavior. At every time step, the desires are then mapped to an actual trajectory by solving an optimization problem with hard constraints, which guarantees safety [22].

Chapter 3

Theoretical background

This chapter gives a brief introduction to Markov decisions processes and reinforcement learning. Notation that is used in subsequent chapters is also introduced. The material in this chapter is based on Kochenderfer [14] and Sutton et al. [27], which provide a comprehensive overview of MDPs and RL.

3.1 Markov decision process

Sequential decision-making problems are commonly modeled as a Markov decision processes. In this framework, an agent chooses an action a, based on the current state s, then receives a reward r, and transitions to a new state s'. An MDP satisfies the Markov property, which assumes that the probability distribution of the next state only depends on the current state and action, and not on the history of previous states. The MDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is a state transition model, R is a reward model, and $\gamma \in [0, 1]$ is a discount factor. The state transition model $T(s' \mid s, a)$ describes the probability that the system transitions to state s' from state s when action a is taken, and the reward model defines the reward of each step as $r = R(s, a, s')$. The goal of an agent is to choose an action at each time step t that maximizes the future discounted return R_t, defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{3.1}$$

where r_{t+k} is the reward at step $t + k$.

In many decision-making problems, the exact state is not known by the agent and it only perceives observations o. A problem with state uncertainty can be modeled as a partially observable Markov decision process, which is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, \gamma)$. Compared to an MDP, the POMDP includes an additional observation space $\mathcal{O}$, and an observation model $O(o \mid s, a, s')$, which describes the probability of observing o in state s', after taking action a in state s.

For many real world problems, it is not possible to represent the probability distributions T and O explicitly. For some solution approaches, only samples are needed, and then it is sufficient to define a generative model G, which samples a new state and observation from a given state and action, i.e., $s' \sim G(s, a)$ for the MDP case, and $(s', o) \sim G(s, a)$ for the POMDP case.

3.2 Reinforcement learning

As mentioned above, the goal of a decision-making agent is to take actions that maximizes the future discounted return R_t. If all elements of the MDP are known, the agent could compute which action that is ideal before executing any actions in the environment, which is referred to as planning. However, in many problems the transition model or generative model is not known to the agent beforehand and it needs to learn how to behave from experience, which is referred to as a reinforcement learning problem. The agent will then act in the environment and observe what happens, in order to figure out a policy π, which defines which action to take in a given state. Figure 3.1 shows a schematic overview of the reinforcement learning problem. The agent is commonly represented by a neural network, which acts as a nonlinear function approximator. Further details on neural networks and how they can be used in RL are described by Sutton et al. [27].

RL algorithms can be divided in model-based and model-free approaches. In the model-based versions, the agent first tries to estimate T and then use a planning algorithm to find a policy. On the contrary, model-free RL algorithms do not explicitly construct a model of the environment to decide which actions to take. The model-free approaches can be further divided into value-based techniques and policy based techniques. Value-based algorithms, such as Q-learning, learn the expected return $\mathbb{E}(R_t)$ of a state in various

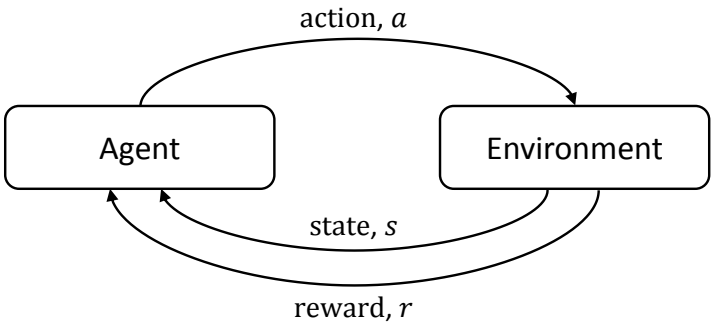


Figure 3.1: *In a reinforcement learning problem, an agent learns a policy π by acting in the environment it should operate in. The agent collects experience by repeatedly taking actions, based on the current state, and then observing the new state of the environment and what reward it receives.*

ways and use this to choose an action, whereas policy based techniques, such as policy gradient methods, learn the policy directly. There are also hybrid techniques that are both policy and value-based, such as actor critic methods.

Genetic algorithms belong to a family of optimization methods that are inspired by the evolutionary mechanisms of natural selection [11]. In general, GAs are suitable for solving optimization problems where the objective function is non-differentiable, or even when an explicit mathematical model does not exist and only a simulation is available. A GA can also solve some versions of RL problems, and is used as an RL method in this thesis.

Chapter 4

Tactical decision-making with RL

Three reinforcement learning approaches to create a tactical decisions making agent for autonomous driving, which were introduced in Paper I, II, and III, are presented in this chapter. Two of the approaches involve model-free RL methods, where the first method is policy based and uses a genetic algorithm (Section 4.3), whereas the second method is value-based and uses a DQN agent (Section 4.4). The transition function is unknown in both of these cases, and little domain knowledge is introduced. In the third method, a simple model of the environment is added, and Monte Carlo tree search and RL are combined to form the decision-making agent (Section 4.5). The three RL methods are henceforth called the GA agent, the DQN agent, and the MCTS/NN agent (where NN refers to neural network). Before the three methods are presented, the test cases, how the simulations were set up, and a commonly used baseline method are described in Section 4.1, and how the decisions making problem is formulated as a POMDP is described in Section 4.2.

4.1 Simulation setup

The three RL methods were applied to different highway driving cases. The main test case of the GA and DQN agents was continuous driving on a three lane, one-way, highway (Figure 4.1a). To indicate the generality of these approaches, the GA and DQN agents were also tested on an overtaking case with oncoming traffic (Figure 4.1b). The MCTS/NN agent was tested on a

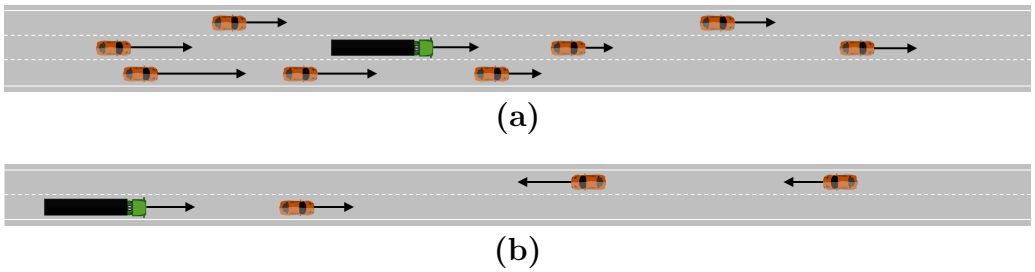
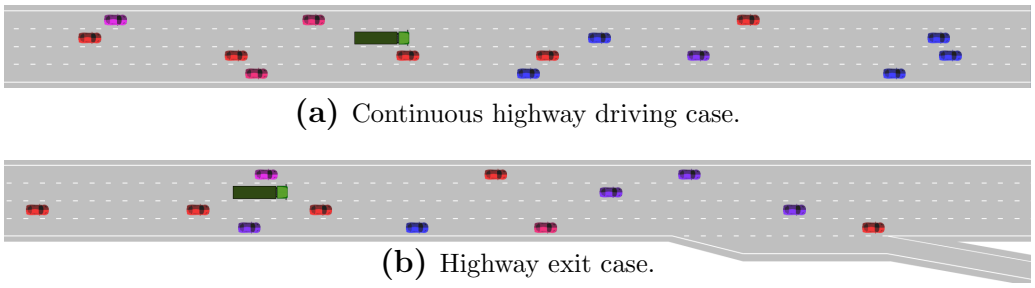


Figure 4.1: *The two test cases that were used to evaluate the GA and DQN agents. Panel (a) shows an example of an initial traffic situation for the one-way highway driving case. Panel (b) shows an example of a traffic situation for a secondary overtaking case with oncoming traffic, displayed* 10 *seconds from the initial state. The ego vehicle, which consists of a truck-trailer combination, is shown in green and black. The arrows represent the velocities of the vehicles.*



(a) Continuous highway driving case.

(b) Highway exit case.

Figure 4.2: *The two test cases that were used to evaluate the MCTS/NN agent. Panel (a) shows an initial state for the continuous highway driving case, whereas panel (b) shows the exit case, when the ego vehicle is approaching the exit on the right side of the road. The ego vehicle is the green truck, whereas the color of the surrounding vehicles represent the aggressiveness level of their corresponding driver models. Red is an aggressive driver, blue is a timid driver, and the different shades of purple represent levels in between. The exact interpretation of aggressiveness level is given in Paper III.*

similar continuous highway driving case, but with four lanes (Figure 4.2a), and on a case when a highway exit had to be reached (Figure 4.2b).

In all the test cases, the surrounding vehicles were randomly generated, with random initial position, speed, and intentions. In order to create interesting traffic situations, the vehicles that were initialized behind the ego vehicle were driving faster than the ego vehicle, and the vehicles that started in front of the ego vehicle were driving slower. The drivers of the surrounding

vehicles were modeled by using the Intelligent Driver Model (IDM) [29] and Minimizing Overall Braking Induced by Lane changes (MOBIL) model [13]. The IDM is a type of adaptive cruise control (ACC) model, that keeps a desired speed when there is no vehicle in front of the vehicle that is being controlled, and otherwise maintains a gap to the preceding vehicle. The MOBIL model is a lane changing model, that makes lane changes with the aim of maximizing the acceleration of all the vehicles that are involved in the traffic situation. A politeness factor controls the balance between the gains and losses of the vehicle that is being controlled and the surrounding vehicles. The combination of the IDM and MOBIL model was also used as a baseline method when evaluating the performance of the different RL agents. Further details on exactly how the simulation environments of the test cases were set up and how the episodes were initialized are given in Paper I, II, and III.

4.2 POMDP formulation

As described in Chapter 3, a sequential decision-making problem can be modeled as a POMDP. This section gives an overview of the POMDP formulation of the decision-making problem in the test cases and for the different approaches, whereas the details are provided in Paper I, II, and III.

- State space $\mathcal{S}$: The state of the system is described by the physical state of all vehicles, i.e., their position and velocity, the state of the driver models of all the vehicles, and a simple road description.

- Action space $\mathcal{A}$: The agents take actions in the longitudinal and lateral direction. The longitudinal actions for the GA and DQN agent consist of vehicle acceleration, whereas the MCTS/NN agent takes decisions on the setpoint of an underlying ACC module. Laterally, the actions consist of staying in the current lane, or changing lanes to the left or right. This changes the setpoint of an underlying path following module, which guides the vehicle to the desired lane.

- Reward model R: Simple reward functions are used, which differ somewhat for the three agents. In short, the agents receive a positive reward that is proportional to the ego vehicle speed, a small negative reward

for changing lanes, and a big negative reward for colliding or driving off the road.

- Transition model T: The transition model is implicitly described by a generative model G. A simple kinematic model is used for the physical dynamics, and the combination of the IDM and MOBIL model controls the surrounding vehicles.

- Observation space $\mathcal{O}$: The observation space consists of the full ego vehicle state, the physical state of the surrounding vehicles, and the road description, i.e., the full state except for the driver model state of the surrounding vehicles.

- Observation model O: A simplified sensor model was used, which could observe the physical state exactly.

4.3 Policy based RL approach, GA agent

Paper I introduces a model-free, policy based, RL approach to tactical decision-making in two different highway cases (Figure 4.1), where a genetic algorithm is used to optimize a structure of rules and actions, and their parameters. The method and the main results are outlined in this chapter, whereas further details are given in Paper I.

4.3.1 Method

A GA with length-varying chromosomes is used to train a rule-based driver model for the two highway cases that were introduced in Section 4.1. A chromosome encodes a set of instructions, which are represented by four genes, $g_1, \ldots, g_4$, described in Table 4.1. Each instruction can encode either a rule or an action. For example, the instruction $[0, 1, 0.2, 0.7]$ would be translated to the rule: *If there is a vehicle in the left lane, in the interval* –60 *m to* 40 *m longitudinally, relative to the ego vehicle, then ...* A chromosome is constructed from a variable set of instruction, which generates a driver

Table 4.1: *Encoding of instructions.*

Gene	Value	Interpretation
g_1	0	Rule: If there is a vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	1	Rule: If there is no vehicle in lane g_2, in the interval g_3 to g_4 longitudinally, relative to the ego vehicle
	2	Action: Change to relative lane g_2, brake or accelerate according to g_3, using pedal level g_4
g_2	–1	Right lane
	0	Current lane
	1	Left lane
g_3	$v_3 \in [0, 1]$	if $g_1 = 0$ or 1, map value v_3 to $[-100, 100]$ m if $g_1 = 2$, g_3 represents braking if $v_3 < 0.5$ and acceleration if $v_3 \geq 0.5$
g_4	$v_4 \in [0, 1]$	if $g_1 = 0$ or 1, map value v_4 to $[-100, 100]$ m if $g_1 = 2$, g_4 represents pedal level

model on the following form:

- *Rule 1*
  - *Rule 2*
    - ▷ *Action 1*
- *Rule 3*
  - ▷ *Action 2*
- ▷ *Action 3*

When the driver model that is generated by the chromosome is to make a decision, it first considers the rules that precedes the first action. If all of them are fulfilled, the first action is executed. If one rule is not fulfilled, the rules that precedes the next action are considered, and so on. If there are no rules associated with an action, as for Action 3 in the example, this action will be executed if no preceding action has been chosen. Decisions were taken at an interval of $\Delta t_{\text{GA}} = 0.1$ s.

In a GA setting, the quality of an individual in a generation is referred to as fitness, which in this case corresponds to the sum of rewards of an

episode. The fitness of an evolved driver model is evaluated in a simulated environment. In short, the agent gets a score of 1 if it completed a 500 m long episode without collisions and at least as fast as the IDM/MOBIL model, and less if it collides or drives slower than the IDM/MOBIL model. If it solves an episode without collisions, it is presented with a new one, up to 500 different episodes. The total fitness of the driver model is defined as the sum of the individual episode scores.

In each GA run, the chromosomes are initialized randomly and the fitness of each individual in the generation are evaluated. Tournament selection with a non-homologous two-point crossover operator is used, which allows the length of the chromosomes to vary over generations. Mutation of the parameters, and inserting or deleting instructions, further diversifies the population. Finally, to guarantee that the performance does not degrade over generations, elitism is used, which means that the best individual of each generation is copied to the next generation without modification.

4.3.2 Results

Three optimization runs were carried out for the one-way highway driving case, with different random seeds of the GA. The fitness of the best individual of each generation is shown in Figure 4.3. After 1,500 generations, which corresponds to around 100,000 hours of driving, all of the three GA runs solved all the 500 test episodes. The final driver models were then applied to 500 new test episodes, which were different from the ones the agents saw during training, and solved all of them without collisions.

The final decoded driver model from one of the GA runs is shown in Table 4.2. In principle, the evolved driver model generates a behavior where the vehicle stays in its lane and accelerates if no vehicle is close in front of it. If a vehicle is present, but there is no vehicle in the left lane, it changes lanes to the left lane. If also the right lane is occupied, it stays in its own lane and brakes. Otherwise it changes to the right lane. This behavior resembles a gap acceptance model [1].

The same method was applied to the overtaking case, with the only difference being a slightly modified fitness function. All test episodes, and an additional 500 unseen episodes, were solved for this case too, but it required around four times as many generations.

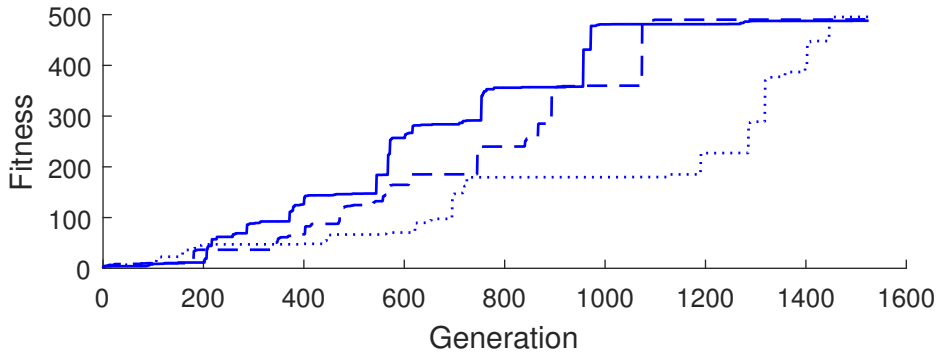


Figure 4.3: *Fitness variation of the best individual in the population, for three optimization runs with different random seeds.*

Table 4.2: *Evolved driver model.*

- If no vehicle in ego lane is within $[-3.4, 21.5]$ m
  - If no vehicle in right lane is within $[63.3, 99.7]$ m
    - ▷ Keep lane, accelerate with pedal level 0.94
- If no vehicle in left lane is within $[-17.8, 77.2]$ m
  - ▷ Do lane change to the left, accelerate with pedal level 0.97
- If no vehicle in ego lane is within $[-2.2, 38.1]$ m
  - ▷ Keep lane, accelerate with pedal level 1.00
- If vehicle in right lane is within $[-18.1, 40.9]$ m
  - ▷ Keep lane, brake with pedal level 0.88

▷ Do lane change to the right, accelerate with pedal level 0.78

- If vehicle in left lane is within $[-75.7, 46.6]$ m
  - ▷ Keep lane, brake with pedal level 0.88

▷ Do lane change to the left, accelerate with pedal level 0.86

4.4 Value-based RL approach, DQN agent

Paper II introduces a model-free, value-based, RL approach to tactical decision-making, applied to two different highway cases (Figure 4.1). The DQN algorithm [18] is used to train a neural network to estimate the expected value of the different actions for a given state. This section outlines the method and the main results, whereas the details on this study are described in Paper II.

4.4.1 Method

Q-learning [34] is a model-free and value-based branch of reinforcement learning, where the objective of an agent is to learn the optimal state-action value function $Q^*(s,a)$, which is defined as the expected return when taking action a from state s and then following the optimal policy π^*, i.e.,

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]. \tag{4.1}$$

The optimal state-action value function follows the Bellman equation,

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right], \tag{4.2}$$

which can intuitively be understood by the fact that if Q^* is known, the optimal policy is to select the action a' that maximizes the expected value of $Q^*(s',a')$.

In the DQN algorithm, a neural network with weights θ is used as a function approximator of the optimal state-action value function, $Q(s,a;\theta) \approx Q^*(s,a)$ [18]. The weights of the network are adjusted to minimize the error in the Bellman equation, typically with some stochastic gradient descent algorithm. Mini-batches with size M of experiences, $e = (s,a,r,s')$, are drawn from an experience replay memory, and the loss function is calculated as

$$L(\theta) = \mathbb{E}_{\mathrm{M}}\Big[(r + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta))^2\Big]. \tag{4.3}$$

A few additional modifications to the basic DQN algorithm, described above, make the learning more stable. See Paper II for the details of the implementation in this study.

Two different DQN agents are tested in this study, which are referred to as Agent1 and Agent2. Both agents use the same state description, which consists of the ego vehicle state, a description of the lanes of the road, and the relative position and speed of the $N_{\max}$ surrounding vehicles. There are m_{ego} states that describe the ego vehicle and the road, and m_{veh} states that describe each of the surrounding vehicles. Agent1 only controls the lane change decisions, whereas Agent2 controls both the speed and the lane changes. An overview of the available actions is given in Table 4.3. Both agents takes decisions every $\Delta t = 1$ s. A positive reward, proportional to the speed, is given for every time step, and a negative reward is given for

Table 4.3: *Action spaces of the two DQN agents.*

	Agent1
a_1	Stay in current lane
a_2	Change lanes to the left
a_3	Change lanes to the right
	Agent2
a_1	Stay in current lane, keep current speed
a_2	Stay in current lane, accelerate with -2 m/s^2
a_3	Stay in current lane, accelerate with -9 m/s^2
a_4	Stay in current lane, accelerate with 2 m/s^2
a_5	Change lanes to the left, keep current speed
a_6	Change lanes to the right, keep current speed

collisions or driving off the road. Additionally, to limit the number of lane changes, a small negative reward is given when choosing the lane changing action.

Two neural network architectures are compared in this study. Both take the state description as an input, and have 3 or 6 output neurons, describing the Q-value for the different actions of Agent1 and Agent2. The first architecture consists of a standard fully connected neural network (FCNN), with two hidden layers. The second architecture introduces a new way of using a temporal convolutional neural network (CNN) structure, which is applied to the part of the input that describes interchangeable objects, in this case surrounding vehicles, see Figure 4.4. The input that describes the surrounding vehicles is passed through CNN layers, which are designed to give the same weights for the inputs of each vehicle, and finally a max pooling layer creates a translational invariance between the vehicles. This structure makes the output independent on the ordering of the vehicles in the input vector, and it also removes the problem of specifying a fixed input vector size, which instead can be made larger than necessary and padded with dummy values for the extra slots. This extra input will be removed in the max pooling layer. Further details on this CNN architecture are explained in Paper II. A general description of CNNs is given by LeCun et al. [15].

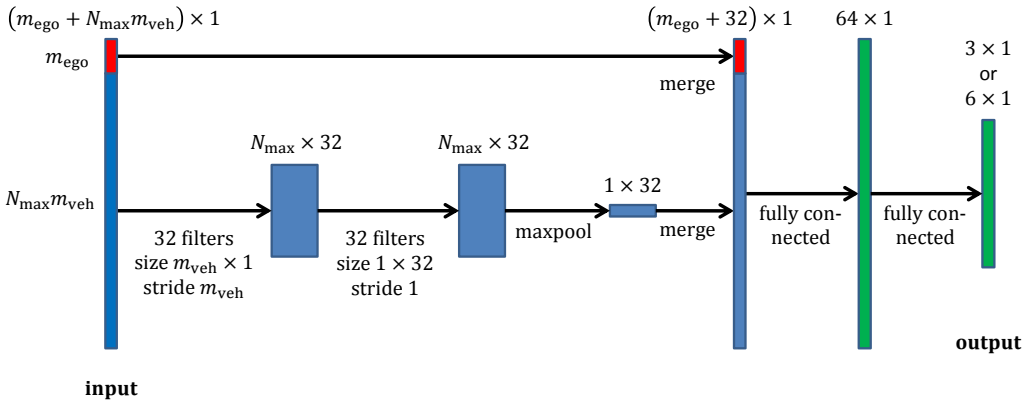


Figure 4.4: *The second network architecture, which uses convolutional neural network layers and max pooling to create a translational invariance between the input from different surrounding vehicles. See the main text for further explanations.*

4.4.2 Results

Five different runs were carried out for the two agent variants and the two network architectures, here called $\text{Agent1}_{\text{FCNN}}$, $\text{Agent1}_{\text{CNN}}$, $\text{Agent2}_{\text{FCNN}}$, and $\text{Agent2}_{\text{CNN}}$. The agents were trained for 2 million training steps, where an experience was added to the experience replay memory and the neural network weights were updated at every step. At every 50,000 training steps, the agents were evaluated on 1,000 random episodes, which were not present during the training.

Figure 4.5 shows the average proportion of successfully completed, collision free, episodes for the four agent variants for the one-way highway driving case. In Figure 4.6, the performance of the agents is compared to the baseline method through a performance index $\tilde{p}$, defined as

$$\tilde{p} = (d/d_{\text{max}})(\bar{v}/\bar{v}_{\text{ref}}), \tag{4.4}$$

where d is the distance driven by the ego vehicle, which is limited by a collision or the episode length d_{max}. The average speed of the ego vehicle is denoted $\bar{v}$, and $\bar{v}_{\text{ref}}$ is the average speed of the ego vehicle when it is controlled by the IDM/MOBIL model.

$\text{Agent1}_{\text{CNN}}$ quickly learned to solve all episodes without collisions (Figure 4.5) by always staying in its lane. Such a behavior results in a situation

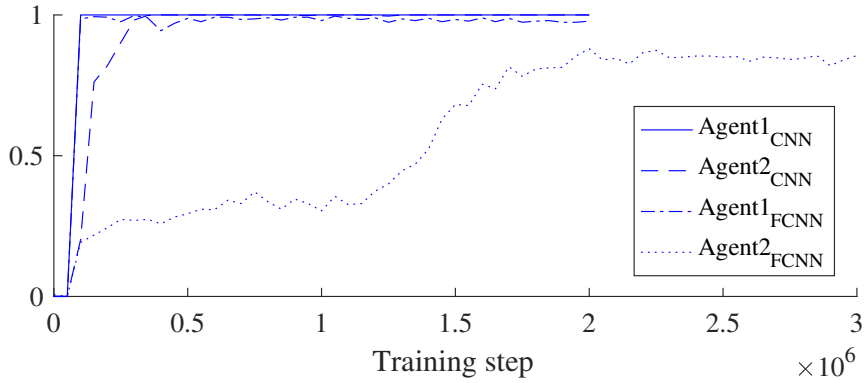


Figure 4.5: *Proportion of episodes solved without collisions by the different agents during training.*

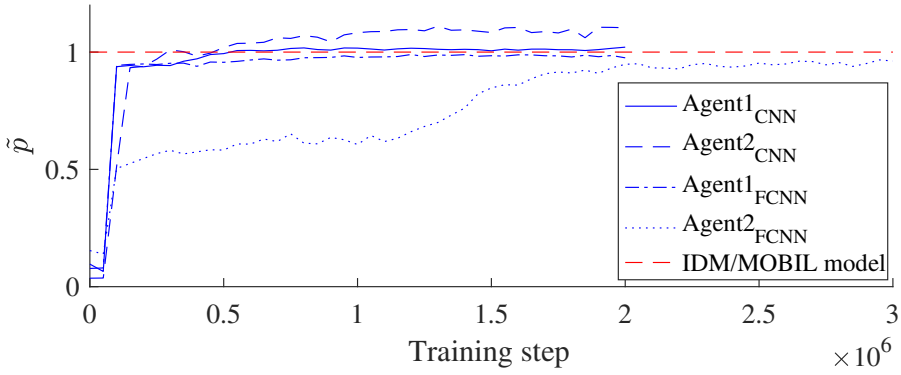


Figure 4.6: *Performance index of the different agents during training.*

where the ego vehicle often gets blocked by slower vehicles and therefore leads to a performance index below 1 (Figure 4.6). However, after around 600,000 training steps, Agent1$_{CNN}$ learned to perform lane changes to overtake slow vehicles, and performed similar to the IDM/MOBIL model.

Agent2$_{CNN}$ learned how to control the speed and make lane changes without collisions in all of the episodes at around 400,000 training steps. At this point, its performance index was on par with the IDM/MOBIL model. With more training, the agent learned even better strategies and after around 1,000,000 training steps, which corresponds to around 300 hours of driving, the performance index stabilized at 1.1.

Table 4.4: *Performance of the DQN agents for the two test cases.*

	One-way highway case		Overtaking case	
	Collision free episodes	Performance index, $\tilde{p}$	Collision free episodes	Performance index, $\tilde{p}_o$
$\text{Agent1}_{\text{CNN}}$	100%	1.01	100%	1.06
$\text{Agent2}_{\text{CNN}}$	100%	1.10	100%	1.11
$\text{Agent1}_{\text{FCNN}}$	98%	0.98	-	-
$\text{Agent2}_{\text{FCNN}}$	86%	0.96	-	-

The agents with the fully connected neural network, $\text{Agent1}_{\text{FCNN}}$ and $\text{Agent2}_{\text{FCNN}}$, performed worse than the CNN agents, both in speed of the training, proportion of collision free episodes, and in performance index. Table 4.4 sums up the results for the four agent variants. It also shows that the CNN agents solved all of the test episodes in the overtaking case, with a better performance than the IDM/MOBIL model.

4.5 Combining planning and RL approach, MCTS/NN agent

Paper III introduces a general framework for tactical decision-making, which combines the concepts of planning and learning, in the form of Monte Carlo tree search and deep RL. This framework is based on the AlphaGo Zero algorithm [24], which is first extended to a more general domain than the game of Go, and then applied to two different highway driving cases (Figure 4.2). This section outlines the method and the main results, whereas the details on this study are described in Paper III.

4.5.1 Method

The decision-making framework of this study uses a neural network f_θ, with parameters θ, to improve the MCTS by guiding the search to the most promising parts of the tree. At the same time, MCTS improves the training process of the neural network by finding long sequences of actions that are

necessary in situations that require a long planning horizon. For each state s, the neural network estimates the value $V(s,\theta)$ and a prior probability $\boldsymbol{p}(s,\theta)$ of taking different actions,

$$(\boldsymbol{p}(s,\theta), V(s,\theta)) = f_\theta(s). \tag{4.5}$$

If $P(s,a,\theta)$ represents the prior probability of taking action a, then $\boldsymbol{p}(s,\theta) = (P(s,a_1,\theta),\ldots,P(s,a_{m_{\text{act}}},\theta))$, for m_{act} possible actions.

The SelectAction function of Algorithm 1 is used to decide which action to take from a given state s_0. Through n iterations, the function builds a search tree, in which the state-action nodes store the set $\{N(s,a), Q(s,a), C(s,a)\}$, where $N(s,a)$ is the number of node visits, $Q(s,a)$ is the estimated state-action value, and $C(s,a)$ contains the set of child nodes. When traversing the tree, the algorithm chooses to expand the action that maximizes the UCB condition

$$UCB(s,a,\theta) = \frac{Q(s,a)}{Q_{\max}} + c_{\text{puct}} P(s,a,\theta) \frac{\sqrt{\sum_b N(s,b)+1}}{N(s,a)+1}, \tag{4.6}$$

where c_{puct} is a parameter that controls the exploration, and $Q_{\max}$ is a normalization parameter.

A progressive widening criterion limits the growth of new state nodes by sampling an old state note if $|C(s,a)| > kN(s,a)^\alpha$, where k and α are parameters that control the width of the search tree. If $|C(s,a)| \leq kN(s,a)^\alpha$, a new state s' is sampled from a generative model $G(s,a)$ of the environment. The new state and reward are added to the set of child nodes and the value of this node is estimated by the neural network as $V(s,\theta)$. Finally, at the end of each iteration, the visit count $N(s,a)$ and Q-values $Q(s,a)$ are updated through a backwards pass.

When the tree search it completed, after n iterations, an action is sampled proportionally to the visit count of the action nodes of the root node

$$\pi(a \mid s) = \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}, \tag{4.7}$$

where τ is a parameter that controls the exploration. During evaluation, the most visited action is greedily chosen, which corresponds to $\tau \to 0$.

Training data is generated from a simulated environment. When an episode ends, after N_s steps, the target values z_i for each step $i = 0, ..., N_s - 1$

Algorithm 1 Monte Carlo tree search, guided by a neural network policy and value estimate.

function SELECTACTION(s_0, n, θ)
  for $i \in 1 : n$
    SIMULATE(s_0, θ)
  $\pi(a \mid s) \leftarrow \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$
  $a \leftarrow$ sample from π
  return a, π
function SIMULATE(s, θ)
  if s is terminal
    return 0
  $a \leftarrow \arg\max_a \left(\frac{Q(s,a)}{Q_{\max}} + c_{\text{puct}} P(s,a,\theta) \frac{\sqrt{\sum_b N(s,b)+1}}{N(s,a)+1} \right)$
  if $|C(s,a)| \leq kN(s,a)^{\alpha}$
    $s' \sim G(s,a)$
    $r \leftarrow R(s,a,s')$
    $C(s,a) \leftarrow C(s,a) \cup \{(s',r)\}$
    $v \leftarrow \begin{cases} 0, & \text{if } s' \text{ is terminal} \\ V(s',\theta), & \text{otherwise} \end{cases}$
    $q \leftarrow r + \gamma v$
  else
    $(s',r) \leftarrow$ sample uniformly from $C(s,a)$
    $q \leftarrow r + \gamma$SIMULATE(s', θ)
  $N(s,a) \leftarrow N(s,a) + 1$
  $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q

are calculated as the received discounted return, according to

$$z_i = \sum_{k=i}^{N_s-1} \gamma^{k-i} r_k + \gamma^{N_s-i} v_{\text{end}}, \tag{4.8}$$

where $v_{\text{end}} = 0$ if s_{N_s} is a terminal state, and otherwise $v_{\text{end}} = V(s_{N_s}, \theta)$. The target action distribution is given by the tree search as $\boldsymbol{\pi}_i = (\pi(a_1 \mid s_i), \ldots, \pi(m_{\text{act}} \mid s_i))$. The tuples $(s_i, \boldsymbol{\pi}_i, z_i)$ are added to a memory, and then the

Table 4.5: *Action space of the MCTS/NN agent.*

a_1	Stay in current lane, keep current ACC setpoint
a_2	Stay in current lane, decrease ACC setpoint
a_3	Stay in current lane, increase ACC setpoint
a_4	Change lanes to the right, keep current ACC setpoint
a_5	Change lanes to the left, keep current ACC setpoint

neural network is trained on the loss function

$$\ell = c_1(z - V(s,\theta))^2 - c_2\boldsymbol{\pi}^\top \log \boldsymbol{p}(s,\theta) + c_3\|\theta\|^2, \tag{4.9}$$

which consists of the sum of the mean-squared value error, the cross entropy loss of the policy, and an L_2 weight regularization term. The parameters c_1, c_2, and c_3 balance the different parts of the loss function. A similar neural network structure as for the DQN agent was used, but in this case with two output heads, which estimate the value and the action distribution of the input state.

As described in Section 4.2, the agent observes the physical state of the surrounding vehicles, but not the driver intentions. A particle filter is therefore used to estimate the parameters of the surrounding drivers, which are assumed to behave according to the IDM/MOBIL model. The most likely state is then used as input to Algorithm 1 and to the generative model.

A state description and reward model that was similar to the DQN agent was used in this study. The action space, given in Table 4.5, is slightly different, where the longitudinal actions changes the ACC setpoint instead of directly controlling the speed. The action space is also pruned at every time step, so that all actions that lead to collisions or driving off the road are removed in the tree search.

4.5.2 Results

The MCTS/NN agent was trained in a simulated environment for the two highway test cases (Section 4.1). An evaluation phase was run at every 20,000 training steps (added training samples), where the agent was tested on 100 random episodes. The average speed $\bar{v}$ of the agent in the evaluation episodes, normalized with the mean speed of the IDM/MOBIL agent $\bar{v}_{\text{IDM/MOBIL}}$, is shown in Figure 4.7, for the continuous highway driving case. The figure also

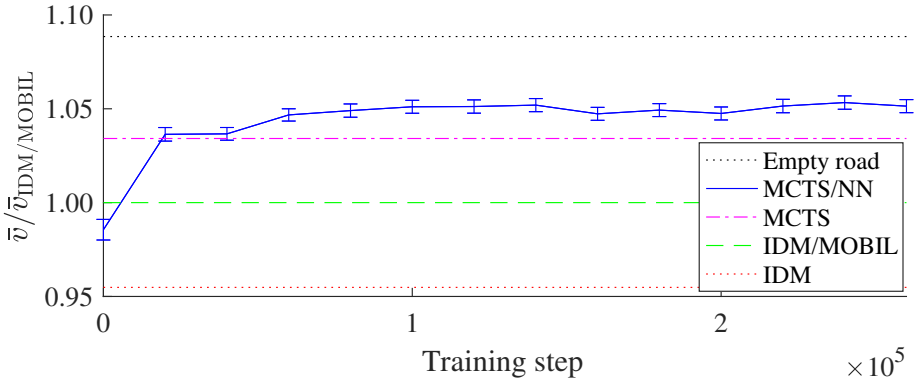


Figure 4.7: *Mean speed $\bar{v}$ during the evaluation episodes for the continuous highway driving case, normalized with the mean speed of the IDM/MOBIL agent $\bar{v}_{\mathrm{IDM/MOBIL}}$. The error bars indicate the standard error of the mean for the MCTS/NN agent, i.e., $\sigma_{\mathrm{sample}}/\sqrt{100}$, where σ_{sample} is the standard deviation of the* 100 *evaluation episodes.*

indicates the average speed when using standard MCTS, referred to as the standard MCTS agent, and when using only the IDM, which always stays in its original lane and can therefore be seen as a minimum performance. The figure shows that the standard MCTS agent outperforms the IDM/MOBIL agent, which does not do any planning. The MCTS/NN agent quickly learns to match the performance of the MCTS agent, which it surpassed after 60,000 training steps.

The highway exit case has a pass or fail outcome, and is therefore conceptually different from the continuous highway driving case. Figure 4.8 shows the proportion of episodes where the exit was reached during the training of the MCTS/NN agent. It quickly learned how to succeed in most episodes, and after 120,000 training steps, which corresponds to 25 hours of driving, it managed to solve all of them. The standard MCTS agent solved 70% and a modified IDM/MOBIL agent solved 54% of the episodes.

A key difference between the compared agents is their planning ability. Figure 4.9 shows a situation where it is necessary to plan relatively far into the future. In this example, the ego vehicle starts 300 m from the exit, six other vehicles are placed in the other lanes, and all vehicles start with an initial speed of 21 m/s. The ego vehicle can only reach the exit by first slowing down and then performing multiple lane changes to the right. This strategy was

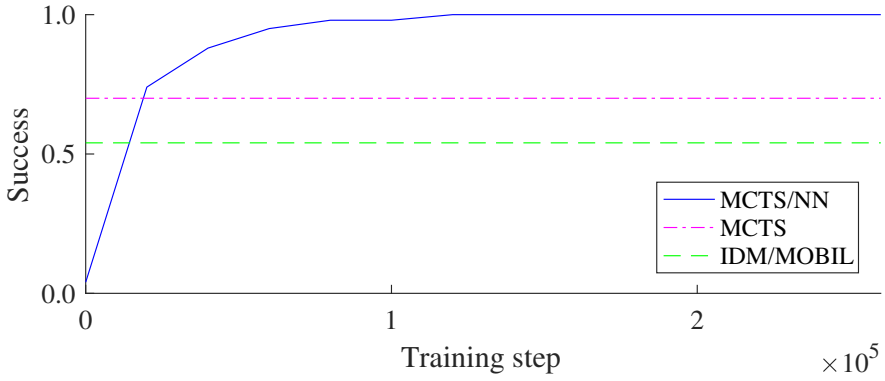
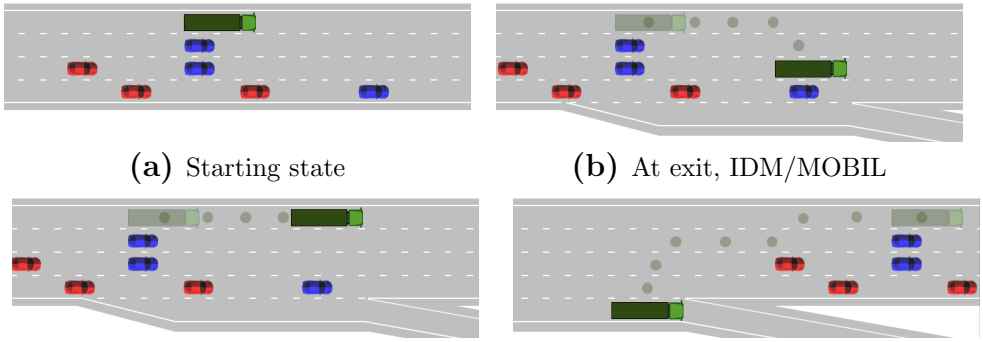


Figure 4.8: *Proportion of successful evaluation episodes, as a function of training steps, for the highway exit case.*



(a) Starting state **(b)** At exit, IDM/MOBIL

(c) At exit, MCTS **(d)** At exit, MCTS/NN

Figure 4.9: *Example of when it is necessary to plan relatively far into the future to solve a specific situation. The initial state,* 300 *m from the exit, is shown in (a), and the state at the exit is shown for the three agents in (b), (c), and (d). The dots show the position of the ego vehicle relative to the other vehicles during the maneuver, i.e., in (b) and (c) the ego vehicle accelerates and overtakes the slower vehicles, whereas in (d), the ego vehicle slows down and ends up behind the same vehicles.*

only found by the trained MCTS/NN agents, whereas the standard MCTS agent did not discover that it can reach the exit and therefore remained in its original lane, to avoid receiving negative rewards for changing lanes. The modified IDM/MOBIL agent tried to accelerate and then change lanes to the right, but was blocked by another vehicle and also failed to reach the exit.

Chapter 5

Discussion

Three different approaches to create a tactical decision-making agent, based on different RL methods, were presented in this thesis. The results show that all methods outperform the baseline IDM/MOBIL model by taking decisions that allows the vehicle to navigate through traffic between 5% and 10% faster. However, the main advantage of the presented methods is their generality and ability to handle driving in different environments, which was demonstrated by applying them to conceptually different highway driving cases. In order to apply the presented methods to a new environment, some domain knowledge is required. A high level state space $\mathcal{S}$ and action space $\mathcal{A}$ need to be defined. Moreover, a reward model R that fulfills the requirements of driving in the new environment also needs to be designed, which is further discussed below. These components are enough to train the DQN agent, while the GA agent requires more domain knowledge in the form of handcrafted features that the GA can build its rule and action structure from. Of the three presented methods, the MCTS/NN agent requires the most domain knowledge, since it needs a generative model G of the environment, a belief state estimator, and possibly knowledge on how to prune actions that lead to collisions.

Table 5.1 shows a summary of the number of simulated driving hours (not execution time) that was necessary to obtain the trained agent for the three different approaches. Although the agents were trained on somewhat different highway driving cases, a qualitative comparison can still be made. The DQN agent requires between two and three orders of magnitude less driving time than the GA agent, which also requires more domain knowledge. Therefore, the DQN agent is advantageous compared to the GA agent, at

Table 5.1: *Required driving time to train the different agents.*

GA agent	100,000 h of driving
DQN agent	300 h of driving
MCTS/NN agent	25 h of driving

least for the test cases considered here. The MCTS/NN agent is even more sample-efficient, and requires one order of magnitude less driving time than the DQN agent to solve the harder exit case. As mentioned in Section 4.5, the planning component of the MCTS/NN improves and guides the training process of the neural network. The pruning of actions that lead to collisions is also likely helping to speed up the training. However, each training sample for the MCTS/NN agent is more computationally expensive to obtain than for the DQN agent, due to the many MCTS iterations that are done for each decision. If the training of the agents is carried out in a simulated environment, where it is relatively cheap to obtain training samples, the importance of the sample efficiency advantage of the MCTS/NN agent can be argued, but if the training is done in real world driving, where each training sample is expensive, sample efficiency becomes important. When assessing the number of driving hours that were needed in the simulated environment, it is important to note that the training episodes were set up in such a way that the agents were frequently exposed to interesting situations, since faster vehicles were initialized behind the ego vehicle and slower vehicles were initialized in front. Real world highway driving often consists of monotone routine driving, which means that training from real driving will likely require more data.

The computational load when deploying the trained GA or DQN agent to make tactical decisions is low. The GA agent needs to check the conditions of its evolved rule structure to decide which action to take, whereas the DQN agent needs to pass the input through the neural network once. The computational load when using the MCTS/NN agent is higher, due to the many MCTS iterations. Every iteration needs to traverse through the search tree, use the generative model once to sample a new state at a leaf node, and query the neural network for the prior probabilities and the value of the leaf node. However, the MCTS/NN agent is anytime capable, i.e., it can return a result after any number of iterations. Even a single iteration, which returns the policy of the neural network, will in many cases give a reasonable result.

The result will in general improve when more iterations are used, up to limit, and the number of iterations that are necessary varies with the complexity of the environment and the traffic situation. See Paper III for more details on the effect of using different number of MCTS iterations.

Paper II introduces a novel way of applying a CNN architecture to state input that describes interchangeable objects. A similar CNN architecture is also used in Paper III. Figure 4.6 and Table 4.4 show that this architecture is advantageous compared to a FCNN architecture, mainly because it solves all the test episodes without collisions, compared to 2% collisions for the FCNN architecture. The CNN architecture also reaches a higher performance index and requires less training. Furthermore, the CNN architecture makes the output of the network independent of the ordering of the objects in the input vector, and it removes the need to specify a fixed input vector size for a fixed number of objects.

As mentioned in Section 4.2, simple reward functions were used for all the three presented approaches. Naturally, the design of the reward model has a strong effect of the resulting driving behavior of the agent. A simple reward model proved to work satisfactory in the cases considered here, but additional aspects, such as the effect on the surrounding vehicles, energy efficiency, and comfort could be included. A reward function that mimics human preferences could be determined by using inverse reinforcement learning techniques [2].

Although the three approaches that were presented in this thesis are conceptually general, an important remark is that when using any of them to produce a tactical decision-making agent, the resulting agent will only be able to handle the type of situations that it was exposed to during the training process. Therefore, it is crucial to set up the training episodes to cover the range of situations of the intended driving case. Furthermore, it is hard to guarantee functional safety when using machine learning techniques to create a decision-making agent. This problem is commonly solved by applying an underlying safety layer, which verifies that the planned trajectory is safe before sending it to the vehicle control system [32].

Chapter 6

Conclusions and future work

The results of this thesis show that it seems promising to use a learning-based approach to create a general tactical decision-making agent for autonomous driving. Three different types of RL methods, which use different amounts of domain knowledge, were analyzed and all of them outperformed a heuristic baseline model in different highway driving cases from a time efficiency perspective. A model-free and value-based RL approach, in the form of a DQN agent, requires little domain knowledge, a few hundred hours of simulated driving to train, and executes well below real time. The training process when combining RL and MCTS is more sample-efficient than the DQN agent, and the resulting agent can solve situations where a long planning horizon is required, but the method uses more domain knowledge and requires more computational power to execute. This implies that the common trade-off of speed vs. generality applies both in the training and the execution of different learning algorithms. Furthermore, for the two methods that use a neural network, a CNN architecture that is applied to a high level state description of interchangeable objects reduces the required training time, increases the time efficiency of the driving, and eliminates all collisions in the test episodes.

The approaches to tactical decision-making that were presented in this thesis are all in an early research stage, and more investigations are necessary to decide which method that seems most promising to develop further and deploy in a real vehicle. All methods could be refined by for example testing different state representations and action spaces, varying the parameters, changing the neural network architecture, and trying more advanced generative models. It would be interesting to test the generality of the differ-

ent methods by applying them to other driving cases, such as intersections and other urban environments. The concept of combining planning and RL seems promising, and additional ways of doing this for the autonomous driving domain could be considered. Another way to improve and speed up the learning could be to incorporate human demonstrations to focus the exploration during the training process [10]. As briefly discussed in Chapter 5, it would also be interesting to use real data to learn and apply a human-like reward function.

Bibliography

[1] K. I. Ahmed, *Modeling Drivers' Acceleration and Lane Changing Behavior*, PhD thesis, Massachusetts Institute of Technology, 1999.

[2] S. Arora and P. Doshi, *A survey of inverse reinforcement learning: Challenges, methods and progress*, CoRR, abs/1806.06877 (2018).

[3] A. Bacha et al., *Odin: Team VictorTango's entry in the DARPA urban challenge*, Journal of Field Robotics, 25 (2008), pp. 467–492.

[4] H. Bai, D. Hsu, and W. S. Lee, *Integrated perception and planning in the continuous space: A POMDP approach*, International Journal of Robotics Research, 33 (2014), pp. 1288–1302.

[5] S. Brechtel, T. Gindele, and R. Dillmann, *Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs*, in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2014, pp. 392–399.

[6] C. Browne et al., *A survey of Monte Carlo tree search methods.*, IEEE Transactions on Computational Intelligigence and AI in Games, 4 (2012), pp. 1–43.

[7] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, *Continuous upper confidence trees*, in Learning and Intelligent Optimization, 2011, pp. 433–445.

[8] F. Damerow and J. Eggert, *Risk-aversive behavior planning under multiple situations with uncertainty*, in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2015, pp. 656–663.

[9] D. J. Fagnant and K. Kockelman, *Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations*, Transportation Research Part A: Policy and Practice, 77 (2015), pp. 167–181.

[10] T. Hester et al., *Deep Q-learning from demonstrations*, in AAAI Conference on Artificial Intelligence, 2018, pp. 3223–3230.

[11] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[12] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura, *Navigating intersections with autonomous vehicles using deep reinforcement learning*, CoRR, abs/1705.01196 (2017).

[13] A. Kesting, M. Treiber, and D. Helbing, *General lane-changing model MOBIL for car-following models*, Transportation Research Record, 1999 (2007), pp. 86–94.

[14] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*, MIT Press, 2015.

[15] Y. LeCun, Y. Bengio, and G. E. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[16] T. P. Lillicrap et al., *Continuous control with deep reinforcement learning*, CoRR, abs/1509.02971 (2015).

[17] J. A. Michon, *A critical view of driver behavior models: What do we know, what should we do?*, in Human Behavior and Traffic Safety, L. Evans and S. R.C., eds., Springer US, Boston, MA, 1985, pp. 485–524.

[18] V. Mnih et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[19] M. Montemerlo et al., *Junior: The Stanford entry in the urban challenge*, Journal of Field Robotics, 25 (2008), pp. 569–597.

[20] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, *Tactical decision making for lane changing with deep reinforcement learning*, in NIPS Workshop on Machine Learning for Intelligent Transportation Systems, 2017.

[21] P. Nilsson, L. Laine, N. van Duijkeren, and B. Jacobson, *Automated highway lane changes of long vehicle combinations: A specific comparison between driver model based control and non-linear model predictive control*, in International Symposium on Innovations in Intelligent SysTems and Applications (INISTA), 2015, pp. 1–8.

[22] S. Shalev-Shwartz, S. Shammah, and A. Shashua, *Safe, multi-agent, reinforcement learning for autonomous driving*, CoRR, abs/1610.03295 (2016).

[23] D. Silver et al., *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, CoRR, abs/1712.01815 (2017).

[24] D. Silver et al., *Mastering the game of Go without human knowledge*, Nature, 550 (2017), pp. 354–359.

[25] E. Sonu, Z. Sunberg, and M. J. Kochenderfer, *Exploiting hierarchy for scalable decision making in autonomous driving*, in IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 2203–2208.

[26] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, *The value of inferring the internal state of traffic participants for autonomous freeway driving*, in American Control Conference (ACC), 2017, pp. 3004–3010.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, second ed., 2018.

[28] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, *Learning negotiating behavior between cars in intersections using deep Q-learning*, in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 3169–3174.

[29] M. Treiber, A. Hennecke, and D. Helbing, *Congested traffic states in empirical observations and microscopic simulations*, Physical Review E, 62 (2000), pp. 1805–1824.

[30] S. Ulbrich et al., *Towards a functional system architecture for automated vehicles*, CoRR, abs/1703.08557 (2017).

[31] S. Ulbrich and M. Maurer, *Towards tactical lane change behavior planning for automated vehicles*, in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2015, pp. 989–995.

[32] S. Underwood, D. Bartz, A. Kade, and M. Crawford, *Truck automation: Testing and trusting the virtual driver*, in Road Vehicle Automation 3, G. Meyer and S. Beiker, eds., Springer, 2016, pp. 91–109.

[33] C. Urmson et al., *Autonomous driving in urban environments: Boss and the urban challenge*, Journal of Field Robotics, 25 (2008), pp. 425–466.

[34] C. J. C. H. Watkins and P. Dayan, *Q-learning*, Machine Learning, 8 (1992), pp. 279–292.

[35] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, *Optimal trajectory generation for dynamic street scenarios in a Frenét frame*, in IEEE International Conference on Robotics and Automation, 2010, pp. 987–993.